Basic IOStreams

Logistics

• Project
  – Part 2 (water) due Sunday, Oct 16th
  • Feedback by Monday
  – Part 3 (block) due Sunday, Oct 30
• Questions?

Exam

• Exam 2
  – Thursday, October 27
  – More details on Thursday.

Logistics

• Final exam
  – Good news…but bad news
  – Good news
    • Last day of finals, November 18th
  – Bad news
    • 8am-10am
  – Room
    • 01-3338

Plan for this week

• I/O Week
  – Today: IOStreams 1
  – Tomorrow: IOStreams 2
  – Thursday: IOStreams 3

IOStreams

• Suite of classes for performing I/O in C++
• Reading and Writing:
  – Standard input and output
  – File input and output
  – Input and Output to strings
Streams

- Like Java, Basic low level mechanism for I/O is the stream
  - Stream is a sequence of bytes

- Unlike Java, the basic stream in C++ is buffered.
  - Used to increase efficiency
  - When the program writes something, it is put into a buffer in memory.
  - The output doesn't appear on the screen until the buffer is flushed (written)

IOStream class inheritance

- cin is an istream
- cout and cerr are ostrains

Other classes inherit from these and add I/O to/from a device, string, or memory

IStream

- Extraction
  - Input from an istream object is called extraction
  - Pulling characters out of the stream and into your program
  - istream manages format conversion
  - istream maintains an error state
    - EOF reached
    - Format errors
    - Serious errors
  - Two types of extraction
    - Formatted/Unformatted

Formatted Extraction

- Done via operator >>
  
```cpp
int i;
icin >> i;
```
- operator>> overloaded for different datatypes.
  - C++ provides operator>> for basic datatypes
  - You can write your own for classes
  - operator>> returns a reference to an istream

```cpp
int i, j, k;
icin >> i >> j >> k;
```
Formatted Extraction

- The error state
  - Three error bits
    - `eofbit` – end of file was reached
    - `failbit` – an extraction failed (format error, premature EOF)
    - `badbit` – the stream is bad and probably can’t be used any more

- Testing the error state:
  - `cin.good()`
    - Returns true if none of the error bits are set
  - `cin.eof()`
    - Returns true if `eofbit` is set
  - `cin.fail()`
    - Returns true if `failbit` or `badbit` is set
  - `cin.bad()`
    - Returns true if `badbit` is set

- Testing the error state.
  - istream overrides some operators to allow an istream to be tested as a boolean:
    - `if ( cin )` is the same as `if( !cin.fail() )`
    - `if( !cin )` is the same as `if( cin.fail() )`
  - One can also write
    - `if( cin >> i )`
    - Which is equivalent to
    - `cin >> i;
      if( !cin.fail() )`

- About EOF
  - EOF is not set when you’ve read the last character from the stream
  - it is set when you try to read one character past the end.
  - If EOF is encountered while skipping leading white space, `failbit` is set.

- Example
  - `sp sp 123 nl`
    - `cin >> i;` // all is fine
  - `sp sp 123`
    - `cin >> i;` // extraction success but `eofbit` will be set…next read will fail

- Must test stream after extraction
  - A good stream does not guarantee more input
    - `cin >> i;
      while( !cin.fail() ) {
        // process the value in i
        cin >> i;
      }
    
    Or
    - `while( cin >> i ){
        // process the value in i
      }`
Formatted Extraction

- But not this:

```c++
while( cin ){
    cin >> i;
    // process the value in i
}
```

Formatted Extraction

- Extraction of `int` datatypes
  - Rules:
    - If the value begins with 0, it is assumed to be an octal number.
    - If the value begins with 0x or 0X, it is assumed to be a hexadecimal number.
    - Otherwise, the value is assumed to be a decimal number.

- You can force a particular conversion by
  - Setting the format flag in your `ios_base` (`ios:dec, ios:hex, ios:oct`)
  - Send a manipulator (dec, hex, oct) to the istream

Example:

```c++
char slash;
int month, day, year;
cin >> month >> slash >> day >> slash >> year;
```

- Use instead:

```c++
cin >> dec >> month >> slash >> day >> slash >> year;
```

Questions?

Unformatted Extraction

- Allows for reading of byte data directly
- Formatted vs. Unformatted
  1. No conversion is done in unformatted extractions: bytes are copied, that is all.
  2. Leading white space is not skipped.
  3. Calls are made to member functions rather than overloaded operators.
     1. `get()`, `getline()`, `read()`

Unformatted Extraction

- Read one character at a time

```c++
ch = cin.get();
while( ch != EOF ){
    // process the character
    ch = cin.get();
}
```

OR

```c++
while( cin.get( ch ) ){
    // process the character
}
```
Unformatted Extraction

• Read multiple chars

```c
#define BUFLEN ...
char buffer[ BUFLEN ];
// reads up to the first , or until BUFLEN chars
cin.get( buffer, BUFLEN, ',' );

OR

// get chars a line at a time
while( cin.getline( buffer, BUFLEN ) ){
  // process the line in buffer
}
```

Unformatted Extraction

• Reading raw bytes

```c
#define N_VALUES ...
char values[ N_VALUES ];
cin.read( values, N_VALUES ); // bytes are read into char arrays
```

Questions?

OStream

• Insertion
  – Output to an ostream is called an insertion
  – Output to an ostream is called an insertion
  – ostream manages format conversion
  – ostream maintains an error state, but it's not as important as it is with input

• Two types of extraction
  • Formatted / Unformatted

Formatted Insertion

• Done via `operator <<`

```c
int i = 7;
cout << i;
```

• `operator<<` overloaded for different datatypes.
  – C++ provides `operator<<` for basic datatypes
  – You can write your own for classes
  – `operator>>` returns a reference to an ostream

```c
int i, j, k;
cout  << i << j << k;
```

Formatted Insertion

• Steps of a formatted insertion

1. The stream's error state is checked. If `failbit` or `badbit` are set, the remaining steps are skipped
2. The value is converted to the appropriate characters, possibly padded, and then inserted into the stream.

Formatted Insertion

• Format state is more important than with input
  – The format state consists of
    • a number of flags (default: none set)
    • the fill character (default: ' ') 
    • Precision (default 6)
    • Minimum Width (default 0)
Formatted Insertion

• Format flags

int flags(); // returns flag settings
int flags (int f); // replaces current settings
int setf (int f); // turns on flags
int unsetf (int f); // turns off flags

enum fmt_flags {
    boolalpha = 0x0001,
    dec = 0x0002,
    fixed = 0x0004,
    hex = 0x0008,
    internal = 0x0010,
    left = 0x0020,
    oct = 0x0040,
    right = 0x0080,
    scientific = 0x0100,
    showbase = 0x0200,
    showpoint = 0x0400,
    showpos = 0x0800,
    unitbuf = 0x1000,
    uppercase = 0x4000
};

• Manipulating format state:

char fill (); // returns the fill char
char fill (char c); // sets the fill char
int precision (); // returns the precision
int precision (int p); // sets the precision
int width (); // returns the width
int width (int w); // sets the width

• Manipulators

- cout << dec;
  - Equivalent to cout.setf ( ios::dec );
- cout << oct;
  - Equivalent to cout.setf ( ios::oct );
- cout << hex;
  - Equivalent to cout.setf ( ios::hex );
- cout << flush;
  - Flushes the output buffer
- cout << endl;
  - Ends a line by inserting a newline and flushing the output buffer

- cout << setw( w );
  - Sets the width to w (affects only next <<)
- cout << setfill( c );
  - Sets the fill character to c (affects all <<)
- cout << setprecision( p );
  - Sets the precision to p (affects all <<)
- cout << setiosflags( f );
  - Equivalent to cout.setf ( f );
- cout << resetiosflags( f );
  - Equivalent to cout.unsetf ( f );

• Allows for writing of byte data directly

- For single byte:
  - cout.put (ch);
- For array of bytes:
  - cout.write (buffer, len);

Unformatted Insertion
No wait…there’s more

- Reading/Writing from/to files and strings
  - fstream – I/O to/from files
  - sstream – I/O to and from strings

…but we’ll leave that until tomorrow

- Questions?